(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
7 February 2002 (07.02.2002)

PCT

(10) International Publication Number
WO 02/10929 A1

(51) International Patent Classification[7]: G06F 13/00

(21) International Application Number: PCT/US01/23490

(22) International Filing Date: 26 July 2001 (26.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/221,411 28 July 2000 (28.07.2000) US
09/703,330 31 October 2000 (31.10.2000) US

(71) Applicant *(for all designated States except US)*: REMOTE COMMUNICATIONS INC. [US/US]; 10721 Monaco Street, Littleton, CO 80124 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: CRANSTONE, Peter, J. [US/US]; 10721 Monaco Street, Littleton, CO 80124 (US). KILEY, Kevin, J. [US/US]; Route 1, Box 325 B-1, Bigelow, AZ 72016 (US).

(74) Agents: COPPOLA, Joseph, V., Sr. et al.; Rader, Fishman and Grauer PLLC, Suite 140, 39533 Woodward Avenue, Bloomfield Hills, MI 48304 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 02/10929 A1

(54) Title: SYSTEM AND METHOD FOR SERVING COMPRESSED CONTENT OVER A COMPUTER NETWORK

(57) Abstract: An in-line compression server (24) is disposed between a user-agent (14) such as a web browser and a web server. The compression server (24) compresses requested objects in real-time and transmits them to the user-agent (14). The compression occurs according to a decompression capability of the user-agent (14), thereby reducing the waiting time for the user-agent (14).

SYSTEM AND METHOD FOR SERVING COMPRESSED CONTENT OVER A COMPUTER NETWORK

BACKGROUND OF THE INVENTION

1. Technical Field

The present invention relates generally to computing systems for use with a computer network, and more particularly, to a system and method for serving compressed content over a computer network.

2. Description of the Related Art

The well-known public computer network known as the Internet allows users to gain access to a wide variety of content, such as text, images, music and the like. One of the most common protocols for communicating over the Internet is the Hypertext Transfer Protocol (HTTP), although there are other protocols in widespread use (e.g., FTP). The HTTP protocol is a stateless protocol, and follows a request/response model. A client computer, through a user-agent such as a web browser, sends a request for an object (i.e., a web page) through the Internet to a server. The server transmits a response, which may include the requested object if it is available to the server.

A large number of users access the Internet through the use of an analog modem, which is a relatively low bandwidth device, particularly when the content sought after by the user involves multimedia. The relatively slow access speeds have been a source of great dissatisfaction not only for the users, but for e-tailers and advertisers as well. It is well documented that the longer a user has to wait for a download, the greater the likelihood that he will just "click out". The slow delivery of content is therefore a problem.

There have been attempts to increase the effective speed of content delivery. For example, it is known generally to use compression to reduce the amount of information that has to be transmitted over a fixed connection. For example, some types of analog modems employ data compression (e.g., MNP) to improve point-to-point throughput. The compression has the effect of improving content delivery. In addition, it has also been contemplated generally to allow for the transmission of compressed information through the Internet, as seen by reference to a document entitled "Hypertext Transfer Protocol –

HTTP/1.1", RFC 2616, available at http:///www.w3c.org. The document discloses an "Accept-Encoding:" request-header field, which allows the browser to notify the web server, when making a request, that it can accept in response certain types of encoded content. For example, the content may be encoded using a GZIP compression method. The document also defines a corresponding server header so that a server can notify the web browser that the content being delivered is in a compressed form and should be decompressed. The document, however, does not disclose how content compression might be implemented on the web server.

One approach for a web server system that purports to deliver compressed content is disclosed at http://www.mozilla.org/projects/apache/gzip (hereafter "Mozilla Compression Server" or MCS). The MCS is disclosed as being limited to delivering HTML format (hypertext markup language) files in compressed form, and requires that such files be compressed in advance of the request from the browser being received by the MCS. That is, two versions of every HTML file exist—an uncompressed version and a compressed version. In operation, the MCS is disclosed as redirecting requests for an uncompressed file to an equivalent compressed file, if one already exists. This approach has several shortcomings. First, it creates maintenance problems on the server end, inasmuch as the operator must regularly maintain up-to-date compressed versions of the files, as needed. This approach also requires increased disk capacity to store the equivalent, albeit compressed, files. Additionally, the foregoing system is limited in that it cannot deliver content, in compressed form, that dynamically changes (i.e., only static files that exist in advance of the browser request can be compressed and made available for download). This restriction limits the usefulness of the approach, especially for rapidly changing content (e.g., news). Additionally, the MCS is limited in the types of content that it can deliver (e.g., only HTML).

Accordingly, there is a need for a method and system for fulfilling requests for content over a computer network that minimizes or eliminates one or more of the problems as set forth above.

SUMMARY OF THE INVENTION

The present invention overcomes the shortcomings of the prior art by providing streaming or "on-the-fly" compression of content destined for a user-agent, such as a web

browser. Moreover, the invention is useful in compressing requested content regardless of the media/file type. The invention also provides a system having an architecture that allows new compression algorithms to be easily and quickly "slotted-in", allowing improved performance and feature enhancement on an on-going basis.

A method of providing compressed objects over a computer network is provided. The first step involves identifying a decompression capability of a user-agent. The next step involves monitoring a request for an object made by the user-agent destined for a server, and a corresponding server response. The next step is performed after the monitoring step, and involves generating a compressed object in accordance with the identified decompression capability of the user-agent. Finally, transmitting the compressed object to the user-agent. Through the foregoing, objects need not be compressed in advance of a request for the object by the user-agent.

In a preferred embodiment, the method further includes the step of determining a line speed associated with the user-agent, wherein the step of generating the compressed object includes several substeps. The method to be described in summary fashion below provides a real-time capability of determining whether streaming compression of the requested file will result in a time savings for the user-agent.

The substeps include determining a compressed size of the requested object based on a compression strategy compatible with the identified decompression capability of the user-agent. Next, determining a compression time for compressing the object from an original size to the compressed size. Next, determining a compressed-object transmission time (e.g., from server to user-agent) based on (i) the line speed and (ii) the compressed size of the object. Further, determining an uncompressed-object transmission time based on (i) the line speed and (ii) the original size of the object. The next substep involves comparing (i) the sum of the compression time and the compressed-object transmission time with (ii) the uncompressed-object transmission time. The final substep involves compressing the object when the result of the comparison step indicates a time savings for the user-agent. In one embodiment, compression occurs when the sum defined above is less than the uncompressed-object transmission time.

Other objects, features, and advantages of the present invention will be apparent to one of ordinary skill from the following detailed description and drawings illustrating features of the invention by way of example, but not by way of limitation.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a simplified block diagram view of a preferred compression server in accordance with the present invention;

Figure 2 is a flowchart diagram of an overall method of operation of the server of Figure 1;

Figure 3 is a flowchart diagram of a method for determining an identity of a user-agent in accordance with the present invention;

Figure 4 is a flowchart diagram of a method of determining whether to compress a requested object in accordance with the present invention;

Figure 5 is a simplified state diagram illustrating operation of a finite state processor (FSP) embodiment; and

Figure 6 is a simplified block diagram view of an alternate, compression proxy server embodiment in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to the drawings wherein like reference numerals are used to identify identical components in the various views, Figure 1 is a simplified block diagram of an overall system 10 useful for implementing a system and method for serving compressed content. Figure 1 shows a server computer 12, and a client computer 14 in communications therewith by way of a computer network, such as a public computer network 16, for example, in one embodiment the Internet. Network 16 may employ Transmission Control Protocol/Internet Protocol (TCP/IP), as known. Other networks, for example, an internet or an extranet may also be used. Network 16 may further comprise a satellite network including other wireless communications technologies.

Server computer 12 as illustrated is configured to receive a plurality of requests for text files, images and the like (hereafter "objects") originating from client computer 14, and generating a response message thereto that includes a compressed version of the requested object. Server computer 12 may comprise conventional hardware known to those of ordinary skill in the art, and is illustrated as including (i) a web server 18 executing on an operating system (O/S) 20 in communication with a database 22, and (ii) a compression server 24 operating in accordance with the teachings of the present invention.

Web server 18 is preferably compatible with the Hypertext Transfer Protocol (e.g., HTTP 1.1). Operating system 20 may also comprise conventional components known to those of ordinary skill in the art. In one embodiment, server computer 12 may comprise a conventional PC including a processor, with 128 megabytes of RAM, and a 4 gigabyte SCSI hard drive, O/S 20 may comprise a slackWare Linux operating system software version 2.0.35, web server 18 may comprise the publicly available Apache HTTP-compliant web server software version 1.3.6 and above.

Database 22 is configured to store a plurality of objects that may be requested by a user-agent such as a web browser and provided with a response by web server 18. Requested object types may include a Graphics Information Format Image file (*.GIF), a Hypertext Mark-up Language file (*.HTML), a Joint Photographic Experts Group image file (*.JPG), a Java Class file (*.CLASS), a Java Archive Redundant type file (*.JAR), a Portable Document file (*.PDF), a Windows Audio file (*.WAV), as well as many other file types, as understood by one of ordinary skill in the art.

Compression server 24 is configured to compress content (i.e., requested objects) in real-time in accordance with a compression strategy that is compatible with the decompression capabilities of a remote client computer 14. The compression ensures that bandwidth consumption through network 16, and more particularly the line connection to client computer 14, which is generally the slowest link between client and server, remains at a reduced level. Compression server 24 is configured generally to account for other factors, such as available computing resources at the server, and predetermined performance requirements, to tailor the level of compression in a real-time environment. By providing compressed objects, which are smaller in size, users will experience shorter wait times for web pages and other objects to download and display (even taking into account decompression time), thereby overcoming a problem as set forth in the Background.

Client computer 14 may be any one of a plurality of conventional computing devices, such as, for example only, a personal computer (PC) executing a Windows O/S (e.g., Windows 98 SE, Windows 2000, etc.), a Macintosh computer, a Unix Workstation, Personal Digital Assistant (PDA's) such as a Palm connected-organizer, hand held computers, wireless phones with web access (e.g., WAP enabled), as well as a variety of other computing devices generally. Client computer 14 includes an operating system 26,

an optional decompression client module 28, and a user-agent such as a web browser 30. A user, such as user 32, interacts with client computer 14 in a manner described below.

Browser 30 is an entity that is capable of requesting an object from an HTTP-compliant server, such as web server 18. Other "user-agents" include search engine site-scrapers and "robots" (both not shown). In one embodiment, browser 30 includes a decompression capability built-in. For example, commercially available web browsers, such as Microsoft Internet Explorer, versions 4.71 and above, and Netscape Navigator, versions 4.55 and above (but not Netscape Preview Release version 6.0) include a capability of decompressing files encoded according to a GZIP compression algorithm. Other commercially available browsers may also include the capability of decompressing content-encoded objects. Browser 30 preferably is HTTP 1.1 compliant. Where browser 30 includes at least one decompression capability, optional decompression client module 28 is not required to practice the invention. In another embodiment, however, where browser 30 does not include a decompression capability, client module 28 may be provided as an adjunct to browser 30. Client module 28 provides a plurality of decompression algorithms corresponding to the compression routines available to compression server 24. The decompressers may be public domain, such as, for example only, various public-domain versions of the LZW compression routine, the public-domain HA decompressor for decompressing compressed text or HTML files, as well as others. In addition to conventional components, client module 28 may further include one or more proprietary decompressors, which are compatible with a corresponding proprietary compressor available on compression server 24. As described below, the architecture of server 24 and client module 28 make adding new compression/decompression algorithms straightforward and fast--no change to web server 18 or user-agent 30 is required.

Initially, user 32 inputs a destination web site address (i.e., URL) into browser 30. Browser 30, in turn, issues an HTTP request 34a (e.g., a GET request) across network 16, which may be routed through one or more intermediate nodes, as understood by one of ordinary skill in the art. The HTTP request 34a, for example, in accordance with the HTTP protocol, may request an object (hereinafter "requested object"), which may be any file or data object located in database 22 associated with web server 18. For example, a requested object may be a home page of a publicly accessible web site, which is typically a static file named index.htm or index.html.

Request 34a is then transmitted to compression server 24. The transport of HTTP request 34a may be in accordance with the well known TCP/IP network protocol. Compression server 24, in-turn, forwards the HTTP request 34a to web server 18, as shown as request 34b. Compression server 24 is configured to identify, at this point (i) a decompression capability of the requesting user agent 30 and (ii) a line speed of the client computer 14. Compression server 24 is configured to make the identification, in-part, based on an analysis of information contained in the various request-header fields of request 34a. The line speed is used in making compression decisions. Compression server 24 is further configured to monitor both the request 34a to determine the identity of the object being requested by browser 30, and, later, a corresponding server response by web server 18.

With continued reference to Figure 1, web server 18 is configured to fulfill the request 34b, provided that the usual conditions are met (e.g., that user-agent 30 is allowed to "request" and "receive"; that the requested object is available, etc.). Web server 18 provides a response 36a, which includes the requested object from database 22. In the conventional case, the requested object is uncompressed, or if already compressed, may be amenable to improved compression (e.g., GIF or JPEG files which are already compressed to some level).

Compression server 24 is configured to then generate, after it has monitored and detected the response 36a, a compressed version of the requested object in accordance with the identified decompression capability of browser 30. In a preferred embodiment, compression server 24 is configured to perform the step of generating the compressed object in real-time through implementation of a finite state processor (FSP), as will be described in greater detail below. Through the foregoing, a dynamic compression server is provided, which overcomes the shortcomings of conventional, "static" compression servers, such as that described in the Background. Compression server 24 is configured to transmit the compressed object as part of a response 36b, destined for user-agent 30. Compression server 24, in a constructed embodiment, can service, in real-time, compression requests upwards of 200 times a second.

Figure 2 is a flowchart illustrating, in greater detail, the process established by dynamic compression server 24. The method begins in step 38.

In step 40, compression server 24 identifies user-agent 30. It bears emphasizing that to reliably perform dynamic compression in a client/server transmission protocol environment (e.g., HTTP over TCP/IP on Internet 16), it is important that compression server 24 determine, and respond accordingly, to the capabilities of the requesting user-agent 30. An initial step of identifying capabilities, particularly decompression capabilities, is to ascertain identity of user-agent 30.

Some user-agents will be capable of receiving certain kinds of compressed content and others will not. For example, early versions of Microsoft Internet Explorer do not contain a built-in GZIP decompressor, while later versions do. Correctly accessing the capabilities of the requesting user-agent 30 is important in order (i) to provide an optimal or maximum amount of compression (i.e., size reduction) for any particular request-response, and (ii) to avoid damage that might otherwise be caused by transmitting data in a format (i.e., a compressed format) compatible with the user-agent. For example, under certain configurations, providing compressed objects to a user-agent incapable of decompressing the same may result in the client computer 14 crashing. This is an undesirable result.

In step 42, compression server 24 determines whether the user-agent (as identified) has decompression capabilities. Compression server 24 is configured to make this determination, in-part, based on the outcome of its identification process in step 40. If the user-agent has no decompression capabilities, the method branches to block 44, wherein the compression server 24 enters into a "pass-through" mode of operation. In the pass-through mode of operation, data objects requested from web server 18 are passed through "as-is" by compression server 24 to browser 30. The method then returns to the starting step 38.

If, however, compression server 24 determines in decision block 42 that the requesting user-agent 30 has decompression capabilities, then the method proceeds to step 46.

In step 46, compression server 24 identifies and inventories the particular decompression capabilities available to the user-agent 30. For example, for some of the commercially-available web browsers mentioned above, only one decompression capability may be available (i.e., a GZIP decompressor). However, when client computer 14 is configured to include client decompressor module 28, a plurality of decompression

capabilities may be available to the user-agent 30. This identification is important inasmuch as an increased number of compression/decompression options provides an increased flexibility to compression server 24 in maximizing the compression process, and thus maximizing the size reduction in the file to be transmitted. The compression/decompression capabilities implemented on server 24 and user-agent 30, respectively, are generally lossless type compression approaches. That is, where data is concerned, exact recovery of the requested object at the remote, client computer 14 is required. However, it should be understood that in certain contexts lossy compression schemes may be implemented between compression server 24 and client computer 14. For example only, lossy compression may be acceptable where the requested object itself has already been compressed according to a lossy compression scheme (e.g., for a jpeg image file, *.JPG). The default is lossless.

In step 48, compression server 24 determines a line speed associated with user-agent 30. It is desirable for compression server 24 to operate in a manner such that the requesting user-agent 30 always receives the requested object at the earliest possible time. Therefore, when the total time it would take to convert a requested object to a compressed form using available reduction methods (and transmit to and decompress at the client) is greater than the time it would take to simply forward the requested object "as-is" (i.e., uncompressed), then compression server 24 is adapted to simply forward the requested data object to the user-agent 30.

One factor in determining whether compressing the requested object would result in a time savings to user 32 depends on the user's line speed. Users 32 with extremely fast connections to Internet 16 are not as likely to see a benefit from the compression of requested data objects as other users 32 who have relatively low bandwidth connections to network 16 (i.e., the majority of present-day users connected to the Internet by way of 56 Kbps analog modems). In an environment such as the Internet, there are no direct means to determine line speed. Thus, unlike point-to-point connections, it is difficult if not impossible to determine what the line speed is, and whether compression should be done. This is one reason why a real-time compression engine has not heretofore been implemented for an environment such as the Internet (i.e., the uncertainty in whether compression will help or hurt user wait time). Several approaches to determine line speed are set forth below.

In one embodiment, line speed is simply retrieved from a trusted "user-profile" database (not shown) accessible by server 24. The user-profile database can contain a nominal, fixed (i.e., registered) line speed for a particular transient origin wherein any line speed variances observed to changing line conditions, network conditions, or the like are optionally not taken into account. In an alternate embodiment, however, compression server 24 is configured in such a way that it ascertains the particular transient origin of user-agent 30 and *updates* the internal line speed based on observed, real-time performance. It should be appreciated that due to the inherent unpredictability of packet arrival in an environment such as Internet the use of a statically defined line speed may result in less than optimal operation.

In yet another embodiment, the web server may determine line speed through the use of a standard HTTP "Keep-Alive" request. All modern implementations of the HTTP protocol now permit and allow for the existence of a Keep-Alive request, which when included in the header that is sent from user-agent 30 to web server 18, notifies the server 18 that more than one request will be originating from the same user-agent. Moreover, the server 18 should expect (or permit) to receive another request from the same user-agent 30 immediately following the completion of the current request. If web server 18 is capable of or allowed to permit the Keep-Alive exchange, then it will not terminate the current exchange with user-agent 30 upon fulfillment of the request (i.e., transmission of the requested object, as it would normally due), but rather will wait to receive the expected next request as soon as user-agent 30 can process the previous response and send the next request.

The "Keep-Alive" scenario employed by the HTTP protocol is a mechanism to make the relationship between servers and user-agents appear to behave more like a standard point-to-point "connection-oriented" relationship.

If a server receives a "Keep-Alive" request from a user-agent 30, then the multiple request/response scenario that results can be used to determine the actual user-agent line speed as follows. The server has available information as to how much data was sent during a previous exchange with any particular user-agent. Since the user-agent itself will not submit the next request in a "Keep-Alive" scenario until it has completely received a response from the previous request, then the server can "time" the interval between sending the last response and receiving the next request and determine line speed based on such

time and the amount of data sent. If a plurality of Keep-Alive exchanges are processed in sequence during a particular session, then the ability to accurately determine the line speed for that one user-agent increases geometrically. The server 18 can store the determined line speed in a local user-profile database (not shown), making it available to server 24 for that particular transient connection origin. The compression server may then use that determination as "trusted" information for a predetermined length of time, or, in the alternative, the compression server 24 can be configured to perform any combination of sequenced or random re-evaluations for that connection on any time frame to insure that the stored information is as accurate as possible.

Through the foregoing "learning" process, the compression server will have available a highly accurate line speed rating for any particular user-agent, and, through the examination of multiple sequential "Keep-Alive" request, can also accurately determine the turn-around time for requests and response and subtract the same from the calculated transmission time and achieve a highly accurate transmission speed rating, for example, expressed in Bytes Per Second (BPS).

In still yet another line speed determination embodiment, an HTTP standard "302 redirect" response is employed. This embodiment is particularly useful in line speed determination where the user-agent does not supply or is incapable of supplying the HTTP "Keep-Alive" field in the request-header. Line speed may be determined through a "forced redirection" test on any new user-agent, which can produce the same accurate line speed rating as would normally be determined automatically through the above-described "Keep-Alive" methodology, as described as follows. When the server receives a request from a previously unknown user-agent, and the server sees that the user-agent is not capable of participating in the HTTP "Keep-Alive" exchange, then the server itself can initiate a series of HTTP standard "302 redirect" responses on a "one time only" or infrequent basis to help determine the real-time line speed associated with the user-agent. The "302 redirect" response contains the URL of the last known real location of the requested resource. When the redirect is made, specifying the same location as that originally requested, the requesting user-agent, in-effect, will immediately submit the same request for the same resource to the same server. The foregoing creates a predictable echo turn-around for the resource, which the server can then time and determine with a high degree of accuracy how quickly any particular user-agent is receiving data from the server.

According to this embodiment, the "302 redirect" would only be applied a single time to determine a line speed; however, its use on a periodic basis may yield improved accuracy.

In still yet another line speed determination embodiment, the compression server 24 may be configured to allow user 32 of client computer 14 to update a line-speed profile parameter in a user-profile database, allowing compression server 24 to have updated and accurate information.

It should be understood that even the same client computer/user-agent may have a different access or origin point to the Internet at different times, due to the increasing mobility of computing devices generally, or changing line conditions. For example, a laptop user may access the Internet through a high-speed connection (e.g., 100 Mbps LAN-T1) while in the office, but use an analog dial-up while travelling. Accordingly, it is expressly understood that the foregoing line-speed determinations are *transient* in nature, and are valid for the duration of a particular access by the user 32.

With continued reference to Figure 2, in step 50, compression server 24 is configured to monitor a request for an object from user-agent 30, and the corresponding response from web server 18. The conventional object retrieval process continues. At this point, compression server 24 may use this time to analyze whether the requested object is even a candidate for compression, based on file type, for example. If the requested object is not available to web server 18 (i.e., it is the "end-point locator" for the object), then web server 18 initiates an "object not found" response, the particular code of which is selected according to the engaged protocol.

If the requested object is available to web server 18, then following the retrieval of the requested object from, for example, a local storage medium such as database 22, web server 18 forwards the same via response 36a, destined for user-agent 30, but which passes first to compression server 24. The method then proceeds to step 52.

In step 52, server 24 determines whether the requested object meets predetermined compression criteria. While the processing will be described in greater detail below in connection with Figure 4, the principal test is whether the user-agent will observe a time savings if the object is compressed, as compared to simply passing the object on to the user-agent "as is" (uncompressed).

If the answer to the inquiry in step 52 is "NO", then the method branches to step 54. In step 54, the requested object provided with message 36a is not compressed, and is

passed through as response 36b destined for user-agent 30. Control of the method then proceeds from step 54 to step 38 where the identification process will begin anew (i.e., because HTTP is nominally stateless, no "history" will be carried forward). There are, however, variations understood by those of ordinary skill in the art, such as, for example only, the above-described "Keep-Alive" scenario.

When the answer to step 52 is "YES", however, the method branches to step 56.

In step 56, the requested object is compressed according to the user-agent's decompression capability, or if there are multiple capabilities, the best one. Particularly, if the user-agent has more than one decompressor available to it (and server 24 has the corresponding compressors), then server 24 is configured to analyze the requested object, including its size and content, as well as the line speed, and select the compression algorithm that will provide a maximum or optimal amount of reduction.

The method then proceeds to step 58, wherein the compressed object is transmitted by compression server as response message 36b through the network 16 to client computer 14. Compression server 24, in this regard, is configured to insert into the response-header of response 36b, an HTTP header field "content-encoding: GZIP" where, for example, the requested object has been compressed according to the GZIP compression algorithm. The foregoing inserted field in the response-header notifies the HTTP-compliant user-agent 30 that the object included in response 36b is compressed according to the specified (e.g., gzip) compression method. If the user-agent 30 is configured to include other decompressors that are known to compression server 24, then alternate compression techniques may be specified in the content-encoding field.

Alternatively, if client computer 14 is configured to include the above-described decompression client module 28, then step 58 may not necessarily include an insertion of a field in the HTTP response-header. Rather, server 24 may insert a signature or identification as a prefix to the compressed data object, along with compression information. Client decompression module 28 monitors the incoming information before it is passed to the application, namely the user-agent application 30. If the client module 28 recognizes the predetermined identification (ID) or signature agreed upon in advance between compression server 24 and client module 28, then client decompression module 28 "wakes up", and decompresses the compressed object according to the specified compression technique defined in heading information attached to the compressed object

itself (as opposed to the response-header portion of the response 36b, which nominally complies with a transmission protocol, such as the HTTP protocol). This form of encapsulation or tunneling allows communication between compression server 24 and client module 28.

After step 58, compression server 24 may conduct a standard "clean-up" phase, including, for example only, cleaning up the stack, de-allocating memory, erasing work files, resetting global statics, and the like. Significantly, however, according to another aspect of the present invention, compression server 24 updates tracking profiles, in one embodiment, which records the results of the most recently executed transaction. The results may include actual compression time, compressed file sizes and the like. The foregoing information provides a means for self-tuning and optimizing the operation of server 24.

Figure 3 shows step 40 of Figure 2 (i.e., identify user-agent) in greater detail. Identifying the user-agent provides a means to ascertain its decompression capabilities, if any. It bears emphasizing that while the Internet is implemented generally to support the HTTP protocol, not all components from end-to-end are fully compliant, nor are there any governing bodies with authority to enforce full compliance. This is particularly evident through pervasive "User-Agent:" spoofing. The "user-agent:" header field in the HTTP protocol was designed to provide an identification of the user-agent to the web server. This header field, however, has been misused and abused, rendering its ability to provide correct information unreliable. For example, to have requests for objects accepted by the widest range of web servers possible, many user-agents have resorted to identifying themselves as one widely available type of user-agent (e.g., a Netscape Browser; User-Agent: Mozilla/4.0). This practice has proliferated in the user-agent community to the point where it is now effectively impossible to rely on the user-agent field, ostensibly created for self-identification. Accordingly, it is unreliable for purposes of the present invention to make inferences as to the capabilities of the requested user-agent based on this field alone.

Accordingly, and with reference to Figure 3, compression server 24 is configured to perform at least steps 60, 62, 64, and 66 for user-agent identification. In step 60, compression server defines a first identification parameter based on the information contained in the HTTP standard request-header field "User-Agent:". This may be

accomplished by examining the request-header in request message 34a. The method then proceeds to step 62.

In step 62, compression server is configured to, and performs the step of defining a second identification parameter based on information contained in a request-header field *other than the* "User-Agent:" field. Another field that may be examined by compression server 24 is the "Accept:" field, which is designed to notify the server of the browser's acceptable file types. However, it has been empirically determined that some browsers include unique identifiers in this field that provide a means to identify the browser. The method then proceeds to step 64.

In step 64, compression server 24 determines the identity using the first and second parameters. The method the proceeds to step 66. This step may involve using an empirically determined knowledge base or matrix.

In step 66, compression server 24 determines whether the user-agent has a decompression capability, and if so, what that decompression capability is, based at least in-part on the determined identity. Additionally, the HTTP header may also be examined and used to determine a decompression capability, as known in the HTTP standard. For example, if the user-agent includes "Accept-Encoding: GZIP, deflate" in the request-header, then compression server 24 may, with high probability, determine that the user-agent has a decompression capability. However, there are certain instances where a user-agent, although identifying itself as having a decompression-capability, does not. In these cases the identity as determined can be used to internally disregard the claim made in the request header that it is decompression capable.

Figure 4 illustrates step 52 (determining whether to compress) of Figure 2 in greater detail. In a preferred embodiment, compression server 24 implements the steps of determining whether to compress, and compress by implementation of a finite state processor (FSP). This is described in detail in connection with Figure 5. It should be understood that other approaches (e.g., such as conventional muli-threaded programming) may be used, depending on the available processing power, the performance expectations, etc.

In step 68, compression server 24 determines a compressed size of the requested object. It should be understood that the requested object, when received by compression server by way of message 36a, will also include in a header portion thereof an object length

or size (hereinafter "original size"). The compressed size of the requested object will depend on at least two factors, first, the particular data that makes up the requested object (i.e., known in the compression art as entropy), and second, the particular mechanism or technique used to implement a proposed compression. It should be understood that, at this point in the method, the compression capabilities of the user-agent have already been determined (step 46 of Figure 2). How close an object is from pure entropy may be determined by examining all of the data defining the object, as is known. In some cases, the file type (e.g., html, jpeg) provides an early indication of the probable gains that can be expected from a proposed compression. In a preferred embodiment, however, statistical modeling is employed wherein less than all of the bits of information defining the object are examined to determine an approximate compressed size of the requested object, were it to be compressed according to the user-agent compression capability. Approaches are known in the art, for example statistical modeling, as set forth in M. Nelson, The Data Compression Book, at p. 20 (1991) hereby incorporated by reference. The method then proceeds to step 70.

In step 70, compression server 24 determines a compression time for compressing the requested object from the original size to the compressed size. This may be done through the use of a table wherein each compression approach has associated therewith a corresponding bytes/second compression-time factor. Accordingly, once the original size of the object to be compressed is known, an approximate compression time may be quickly and easily determined.

In step 72, compression server 24 determines a compressed-object transmission time based on line speed and compressed size. This is the time it would take to transmit the compressed file (i.e., a reduce number of bytes) to the user-agent 30.

In step 74, server 24 determines an uncompressed-object transmission time based on line speed and the original size of the object. This time corresponds to the total time it would take until the user-agent would have access to the requested object were it sent "as-is".

In step 76, compression server determines whether compressing the object would result in a time savings for the user-agent. That is, a comparison is made, in a preferred embodiment, between (i) the sum of the compression time, the compressed-object transmission time, and the decompression time at the user-agent, and (ii) the

uncompressed-object transmission time. Compression server 24 also determines whether there is sufficient computing resources to carry out the proposed compression in real-time. This may be done through standard operating system API (application program interfaces). When the results of this comparison indicate that compressing the object will result in a time savings for the user-agent, then compression server compresses the object, as indicated in step 78. If the result of the foregoing comparison indicates that there would be no time savings, then the method branches to step 80, wherein no compression occurs and the object is forwarded to the user-agent "as-is".

Stated another way, if the requested object is already as close to pure entropy as possible, given the constraints of computing resource power (i.e., a practical factor which must always be taken into account) and the time available to reduce it further, and if the format of the requested object is already compatible with what is known about the user-agent's ability to expand and restore the content, then the requested object is simply returned to the user-agent "as-is".

However, if the requested object is not already as close to pure entropy as possible, and there is both sufficient time and computing power to reduce the content further in a manner that is compatible to what is known about the requesting transient user-agent's ability to expand and restore the content, then compression server initiates the process of dynamic compression to reduce the size of the requested object. In most instances, for example, when the user-agent is implemented on conventional PC hardware platform, having mid-level computing power or better, a decompression time at the client computer may not be a limiting factor. However, on a computing resource limited device, such as a PDA (e.g., a Palm connected-organizer), the decompression time becomes extremely significant (due to low computing power) and may be dispositive in deciding not to compress the requested object.

In one embodiment, compression server is configured to operate in one of three modes in deciding whether to compress. A first mode is used when an original file size is below a first threshold. The small files are simply forwarded "as-is". In a second mode, for files having an original size between the first threshold and a second threshold greater than the first threshold, the foregoing method illustrated and described in connection with Figure 4 is used. Finally, for files having a size greater than the second threshold (very large files) compression server 24 operates in a third mode. In the third mode, for very

large files, and typically for slow line speeds, the expected time savings are likely so significant that it makes sense to perform the actual compression of the file according to multiple ones or all of the available compression capabilities, and then determine which compression strategy in-fact provided the highest degree of reduction. It is the smallest file that is then transmitted.

Figure 5 is a simplified state diagram of a finite state processor embodiment of compression server 24. The diagram 200 includes a plurality of states connected by transitions.

Figure 5 shows idle state 202. The method occupies idle state 202 when there are no current client connections. As shown by arrow-headed line 204, server 24, in the idle state 202, may conduct cleanup, maintenance, and logging tasks, all as described in detail above. In addition, idle state 202 is a "return" state following the processing of a client request, and which is configured to perform cleanup that naturally follows the termination of a most recently processed client request. When a request for a new client connection is received by server 24, the process transitions to an initialization state 206. In state 206, a new request structure is allocated, and initialized. The process then transitions to a "target get socket" state 208.

The target entity is that entity, such as web server 18, that will be responsible for retrieving the actual, on-line requested object. In state 208, server 24 contacts the target entity to make sure it is still alive and available. If the primary target entity is not available, then backup targets may be contacted. If there are no targets currently available to fulfill the actual request by user-agent 30, then the user may receive an error. If a valid "socket" is obtained from the target entity that will fulfill the request for an object, then the process proceeds immediately to subsequent states for completion of the target connection.

In target-connect 1 state 210, server 24 initializes the request variables used to complete the connection with the target entity that will fulfill the request for the object by user-agent 30. The process then proceeds to state 212.

Target_connect 2 state 212 is the state that actually completes the connection with the target entity that will fulfill the request by user-agent 30.

Once a "new" connection (or the re-use of an existing connection) to a target entity from server 24 has been successfully established, then the next state initializes the request variables that will be used during the process of accumulating the complete client request

(e.g., request 34a in Figure 1). States 216, 218, and 220, which involve accumulating an HTTP header from a TCP/IP connection, are "re-used" during the response phase when the actual, requested object is being returned from the target entity (e.g., web server 18) to the user-agent 30. These states use a mode flag to know if its current responsibility is to accumulate an initial client request header or a returning response header. As shown in 214, the mode value will as an initial matter be set to accumulate an HTTP header associated with the request, as an initial matter (e.g., a mode value may be set to "zero"). Alternatively, the mode value may be set to a "one" if the accumulation is of a target entity response-header (e.g., mode = response).

In state 218, once the request variables that control the accumulation of either a client request, or a target response-header have been initialized (by state 216), the accumulation process itself is handled by state 218. Primary FSP state 218 is performed by a plurality of substates to assist in the collection of the complete inbound request/response header. In a first substate of state 218, server 24 accumulates HTTP header field content information. In a second substate (one_CR_seen), server 24 is configured to watch for the standard HTTP "end of header" "signal". If seen, then the process advances to the next processing substate. In the third(substate (one_LF_seen), server 24 is configured to watch for the standard HTTP "end of header" signal. If seen, the method advances to the next processing substate.

In the fourth substate (substate two_CR_seen), the watch for the standard HTTP "end of header" signal is cancelled, in as much as the < CR > < LF > "end of header" sequence has been detected. The process then transitions to a Get_Body 1 state 220. Once the entire HTTP request/response header has been accumulated, and it has been determined that body data exists, state 220 is configured to accumulate the actual body data associated with the request/response. When the mode, as described above has been set to accumulate a request header, then the process transitions to state 222 (Evaluate_ Request 1).

State 222 is the FSP state that evaluates an inbound client request and determines the eligibility of the requested object for compression. If it is determined that the requesting user-agent is incapable of accepting any form of compressed data that the server is capable of delivering, or it is determined that the object requested is not a candidate for compression, then state 222 sets passthrough flags for this request, and proceeds directly to states that forward the request to the target entity that will fulfill the request. This is shown

as a transition through steps and states 224 and 226. When the passthrough flags are set, then the states that accumulate the target entity response (i.e., states 216, 218, 220) will pass control directly to the passthrough states 228 and 230 to thereby forward the requested object "as is" to the original requesting user-agent.

State 232 is a special state that is only used during the accumulation of a client request if that request is a standard HTTP POST transaction. This state accumulates all of the post data itself.

State 226 is the FSP state that transmits the original client request to the target entity tasked with the actual fulfillment of the requested object. This state, as shown at 234, changes the internal mode flag to represent the switch from "client request accumulation mode" to "target response accumulation mode" so that accumulation states 216, 218, and 220 can be reused to accumulate the HTTP response header that will be arriving from the target entity.

The process transitions to states 216, 218, and 220 where the returning HTTP response, which contains the requested object, is accumulated. Since the mode has already been set to accumulate a "response", after the response has been acquired, the process transitions to state 236 for evaluation.

Evaluate_response state 236 is specifically configured to evaluate the response header. If the requesting user-agent 30 is capable of receiving compressed data and it is determined that the response object is a candidate for compression, then the process transitions to compression states 238 and 240 for further real-time processing of the response object.

Alternatively, if the requesting user-agent 30 is not capable of receiving compressed data, or it has been determined that the response object is not a valid candidate for compression, then the process transitions to the passthrough states 228 and 230, which deliver the response back to the requesting user-agent 30 "as is". In particular, state 228 is the FSP state that accumulates a response buffer directly from the web server 18 while operating in a passthrough mode. State 230 is the FSP State that transmits a newly received response buffer directly to the original requesting user-agent 30 while operating in a passthrough mode.

State 238 is the FSP state that initializes and validates the request variables that will be used to control the real-time compression of the requested object. If the compression

phase fails or it is determined by compression server 24 that the original requested object should be returned to the user-agent 30, then state 240 is configured to set control flags to that effect, and directly enters state handlers which operate to return the requested object "as is" to the requesting user-agent 30. If the compression phase succeeds, then the process transitions from state 240 transitions to state 242.

State 242 is the FSP state that initializes the request variables that are used to transmit the compressed, requested object to user-agent 30. State 244 is the FSP state that actually transmits the compressed object back to original requesting user agent.

The Internet enjoys an abundance of innovation, and therefore it is foreseeable that better, more efficient compression algorithms will be developed in the future. According to another aspect of the invention, a compression server and client decompression module are provided that together provide for rapid and straightforward "snap-in" of new algorithms.

In accordance with the present invention, a mainline code executing on compression server 24 is configured to be able to load a compression routine from a "shared library", a Dynamically Linked Library in the Microsoft Windows environment (DLLs). When a new compression algorithm becomes available, it need only be configured as a *.DLL. The new *.DLL is then loaded into the operating system of compression server 24, wherein the new compression algorithm is thereafter available.

In an alternate embodiment, any new compression algorithm will likely have a stand-alone command line executable version of both the compressor and the decompressor. Such external compressors and decompressors can be "shelled out" or "called" from the mainline code and benefit from the results of the output. Compression server 24, in one embodiment, has the ability to do this "shell out" at any time, if so instructed in accordance with a configuration file.

Figure 6 illustrates a second embodiment of the present invention, namely a proxy server implemented system 110. Figure 6 shows a system through which a user-agent 30 can receive compressed content from a user-selected one of a plurality of web server computers $112_1, \ldots, 112_n$. The embodiment of Figure 6 has particular utility inasmuch as the user 32 by way of user-agent 30 can obtain the benefit of receiving faster downloads by way of compressed content without any additional requirement on the part of the web

server computers 112. User 32 need only configure user-agent 30 to use a proxy server computer 82.

System 82 is configured generally to receive requests from user-agent 30, designated 88a in Figure 5. System 82 is further configured to deliver compressed content to client computer 14, in response to the request, from a user-selected web server computer $112_1, \ldots, 112_n$. System 82 comprises, functionally, two components: (i) a compression server component, and (ii) a proxy server component, preferably a caching-type proxy server. Although illustrated in Figure 6 as two separate hardware units, it should be understood that both the compression server 24 and the proxy server component may be implemented on one hardware unit, based upon available resources and predetermined target performance levels.

The proxy server component may comprise conventional computing hardware, such as a Pentium-class processor, suitable amounts of memory (RAM, ROM), and hard drive storage. The proxy server component further includes an operating system 83, and proxy cache server software 84. Operating system 83 may comprise conventional operating system software, such as the widely, commercially available UNIX or Windows-based operating systems (e.g., Solaris (Sun Mircro Systems), Linux (various distribution source), and Windows NT (Microsoft Corporation, Redmond, Washington). Proxy cache server software 84 may comprise conventional software known to those of ordinary skill in the art, including, but not limited to, Microsoft Proxy Server version 2.0 (when operating system 83 comprises Microsoft Windows NT, or Windows 2000, all available from Microsoft Corporation, Redmond, Washington), or squid web proxy cache server software version 2.0 (available as a download, from FTP://ftp.squid-cache.org/pub/, as well as a variety of mirror sites), when the operating system 82 comprises one of a plurality of a conventional Unix systems, such as Linux, Free BSD, AIX, and the like.

Likewise, operating system 84 may comprise a UNIX variant, one of the Windows NT variants, for example only, compression sever 24, in the embodiment illustrated in Figure 6, is identical to that described above in connection with Figure 1.

In operation, the user-agent's request flows from client computer 14 to system 82 by way of request message 88a. Compression server 24, via operating system 85 provides the request to proxy cache server 84, indicated as message 88b. If the proxy cache server 84 can fulfill the request out of its own cache, then it does so by a return response 90c.

However, to the extent that the proxy cache server cannot fulfill request 88b, proxy cache server will forward the request to the end-point web server by way of request messages 88c and 88d, as conventional. In a preferred embodiment, according to HTTP, the selected one of the end-point web servers $112_1, \ldots, 112_n$ provides a response, designated 90a. If the web server 112 is able to fulfill the request, it does so by including the requested object as part of the response 90a. If unable to fulfill the request, a response 90a is still made, however an appropriate error message is included, rather than the requested object. The response 90a traverses network 16 and arrives at proxy server 84 as response 90b. Proxy server 84 in-turn, forwards the requested object by way of message 90c. Compression server 24 operates in an identical fashion to the compression server 24 described above in connection with Figure 1. Accordingly, at this point, compression server 24 has already identified a line speed associated with user-agent 30, has identified what, if any, decompression capabilities the user-agent 30 has, and will proceed to evaluate the requested object obtained in response 90c to determine whether compression will occur, all as described above. Compression server 24 generates response 90d which is transmitted through network 16 through client computer 14, particularly user-agent 30, also as described above. The embodiment of Figure 5 has the principal advantage that compressed content can be delivered to a user-agent 30 without any affirmative changes to the web sites provided by web server computers $112_1, \ldots, 112_n$.

It is to be understood that the above-description is merely exemplary rather than limiting in nature, the invention being limited only by the appended claims. Various modifications and changes may be made thereto by one of ordinary skill in the art which embody the principles of the invention and fall within the spirit and scope thereof.

CLAIMS

1. A method of providing compressed objects over a computer network comprising the steps of:

(A) identifying a decompression capability of a user-agent;

(B) monitoring a request for an object made by the user-agent destined for a server and a corresponding server response;

(C) after said monitoring step, generating a compressed object in accordance with the identified decompression capability of the user-agent; and

(D) transmitting the compressed object to the user-agent.

2. The method of claim 1 further including the step of determining, prior to said step of generating said compressed objects, whether transmitting said compressed object will result in a time savings for the user-agent.

3. The method of claim 1 wherein said identifying step includes the substeps of:

defining a first parameter based on information contained in a Hypertext Transfer Protocol (HTTP)-compliant "User-Agent" request-header field;

defining a second parameter based on information indicative of an identity of the user-agent based on information in request-header fields different from the "User-Agent" request-header field; and

determining the identity of the user-agent using the first and second parameters.

4. The method of claim 3 wherein said identifying step further includes the substep of:

selecting a decompression capability based on the identity of the user-agent.

5. The method of claim 1 wherein said monitoring step includes the substeps of:

parsing the request to identify a file type associated with the requested object; and

marking the requested object as a candidate for compression when the identified file type matches at least one of a plurality of preselected file types.

6. The method of claim 1 further including the step of determining a line speed associated with the user-agent, and wherein said generating step includes the substeps of:

determining a compressed size of the requested object based on a compression process corresponding to the identified decompression capability of the user-agent;

determining a compression time for compressing the object from an original size to the compressed size;

determining a compressed-object transmission time based on the line speed and the compressed size of the object;

determining an uncompressed-object transmission time based on the line speed and the original size of the object;

comparing (i) the sum of the compression time and the compressed-object transmission time and (ii) the uncompressed-object transmission time; and

compressing the object when a result of said comparing step indicates a time savings for the user-agent.

7. The method of claim 6 wherein said sum further includes a decompression time at the user-agent to recover the requested object.

8. The method of claim 6 wherein said step of determining a compressed size of the requested object includes the substeps of:

sampling a preselected number of data bits less than all of the data bits of the requested object;

determining an entropy parameter using the preselected number of samples in accordance with a statistical model; and

calculating the compressed size using the entropy parameter.

9. The method of claim 6 further including the step of determining available computing resources, and wherein said step of compressing further includes the substep of:

determining whether the available computing resources will allow compression of the object in the compression time.

10. The method of claim 1 wherein the compression is lossless.

11. The method of claim 1 wherein a client computer on which the user-agent executes includes a client decompression unit, said step of identifying the user-agent comprises the substep of:

transmitting from the client computer a signature identification.

12. The method of claim 1 wherein said steps are implemented on a computer according to a finite state processing (FSP) strategy.

13. A compression server for providing compressed objects over a computer network comprising:

a processor;

a memory;

a first module in said memory executed by said processor configured to identify a decompression capability of a user-agent;

a second module in said memory executed by said processor configured to monitor a request for an object made by the user-agent destined for an object server and a corresponding object server response;

a third module in said memory executed by said processor configured to generate, after said request and response has been monitored, a compressed object according to said decompression capability of said user-agent; and

a fourth module configured to transmit said compressed object to said user-agent.

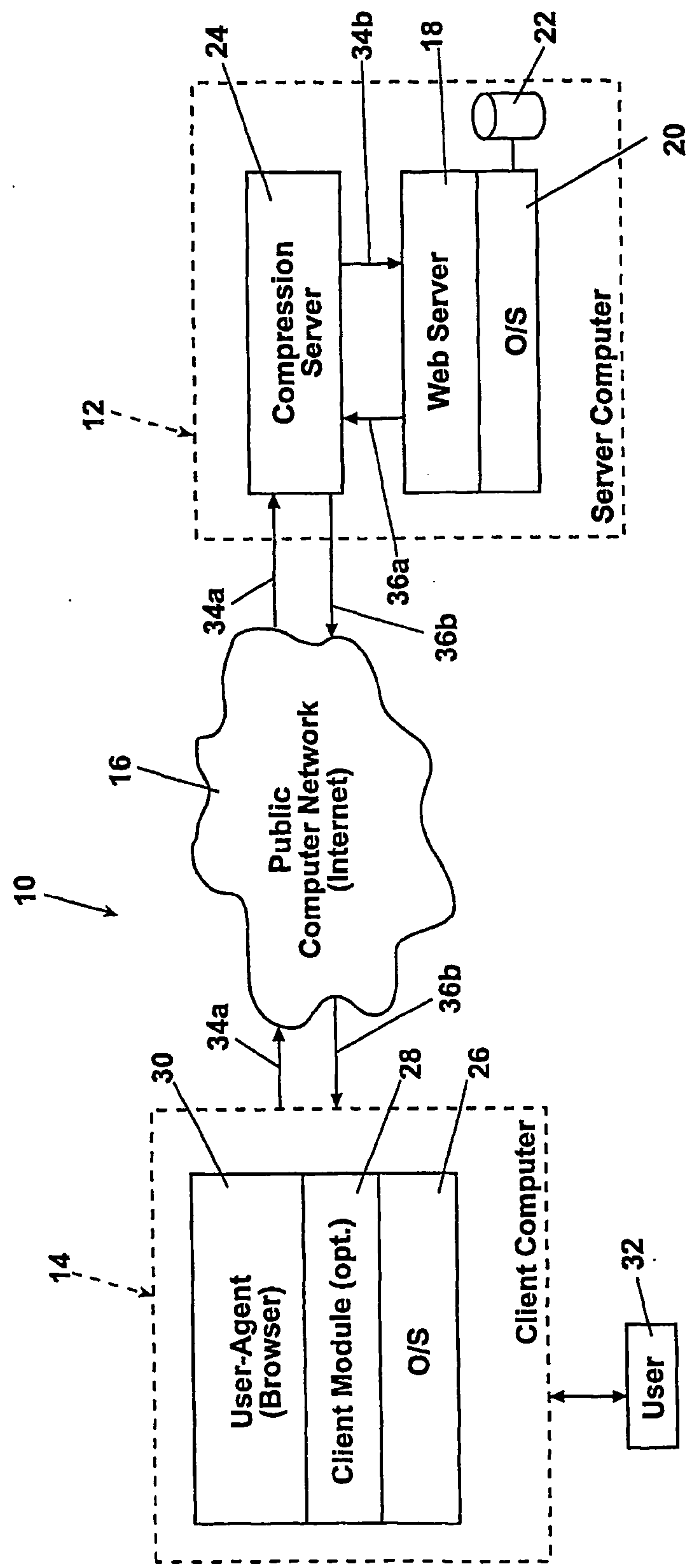
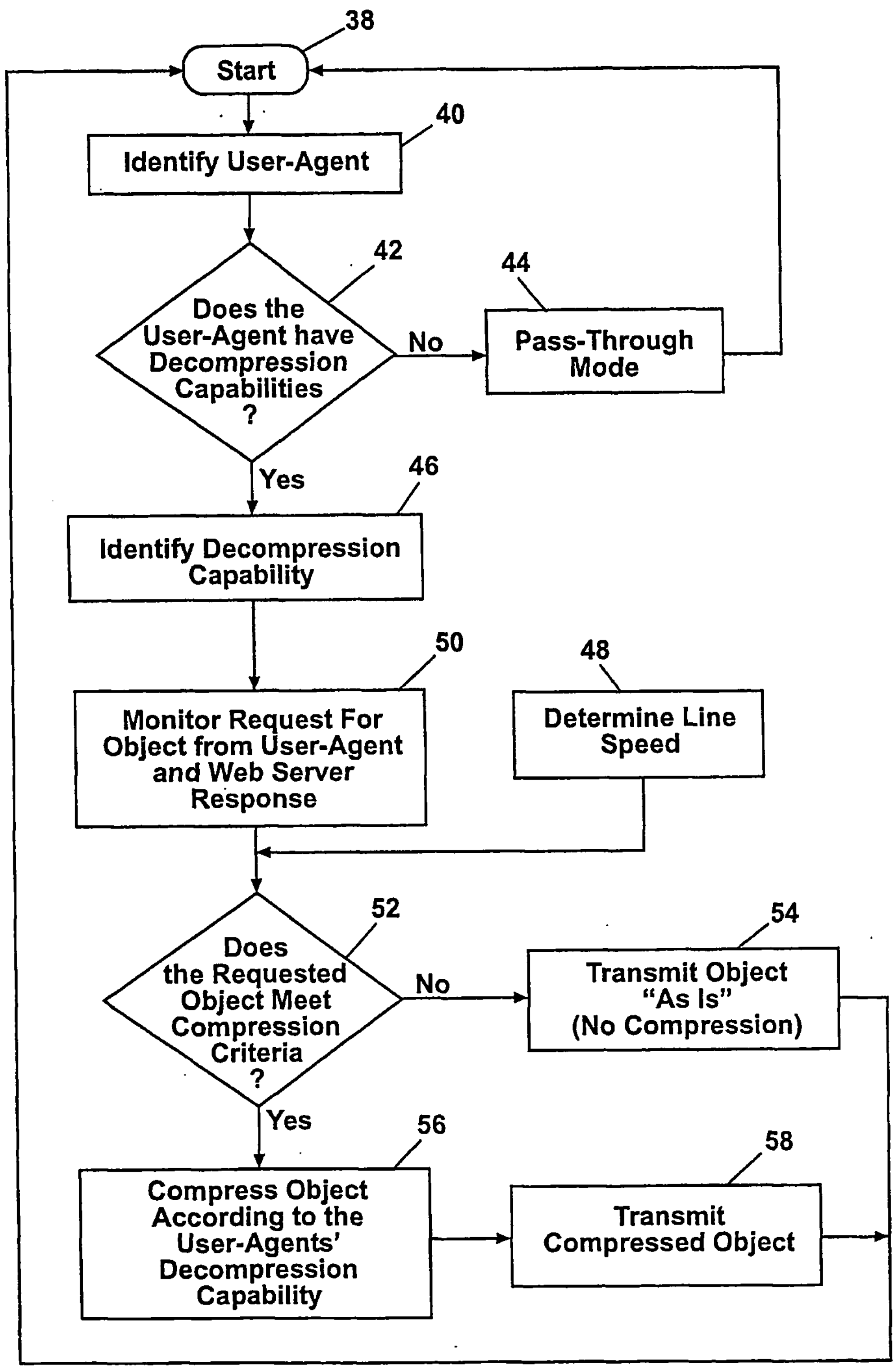

10
12
Compression Server
24
34b
Web Server
18
O/S
20
22
Server Computer
36a
34a
36b
16
Public Computer Network (Internet)
34a
36b
14
User-Agent (Browser)
30
Client Module (opt.)
28
O/S
26
Client Computer
User
32

Fig. 1


38
Start
40
Identify User-Agent
42
Does the User-Agent have Decompression Capabilities ?
No
44
Pass-Through Mode
Yes
46
Identify Decompression Capability
50
Monitor Request For Object from User-Agent and Web Server Response
48
Determine Line Speed
52
Does the Requested Object Meet Compression Criteria ?
No
54
Transmit Object "As Is" (No Compression)
Yes
56
Compress Object According to the User-Agents' Decompression Capability
58
Transmit Compressed Object

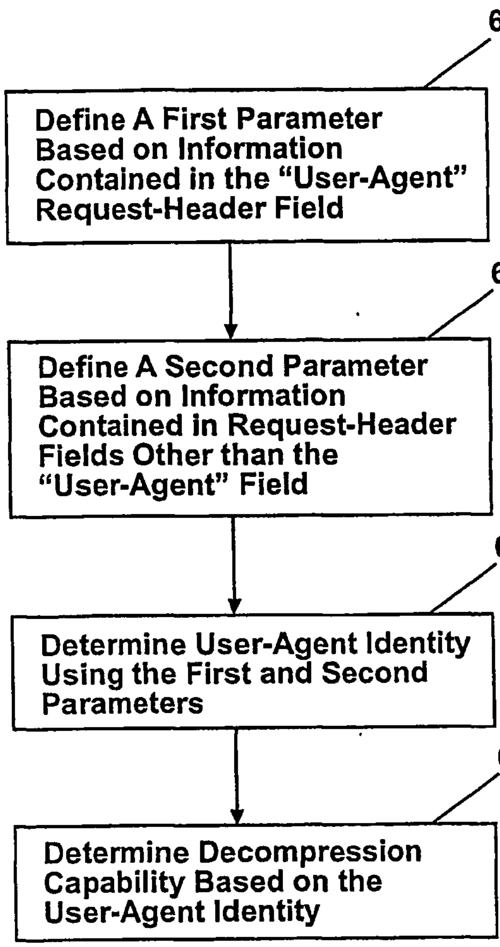

60
Define A First Parameter Based on Information Contained in the "User-Agent" Request-Header Field
62
Define A Second Parameter Based on Information Contained in Request-Header Fields Other than the "User-Agent" Field
64
Determine User-Agent Identity Using the First and Second Parameters
66
Determine Decompression Capability Based on the User-Agent Identity

Fig. 3

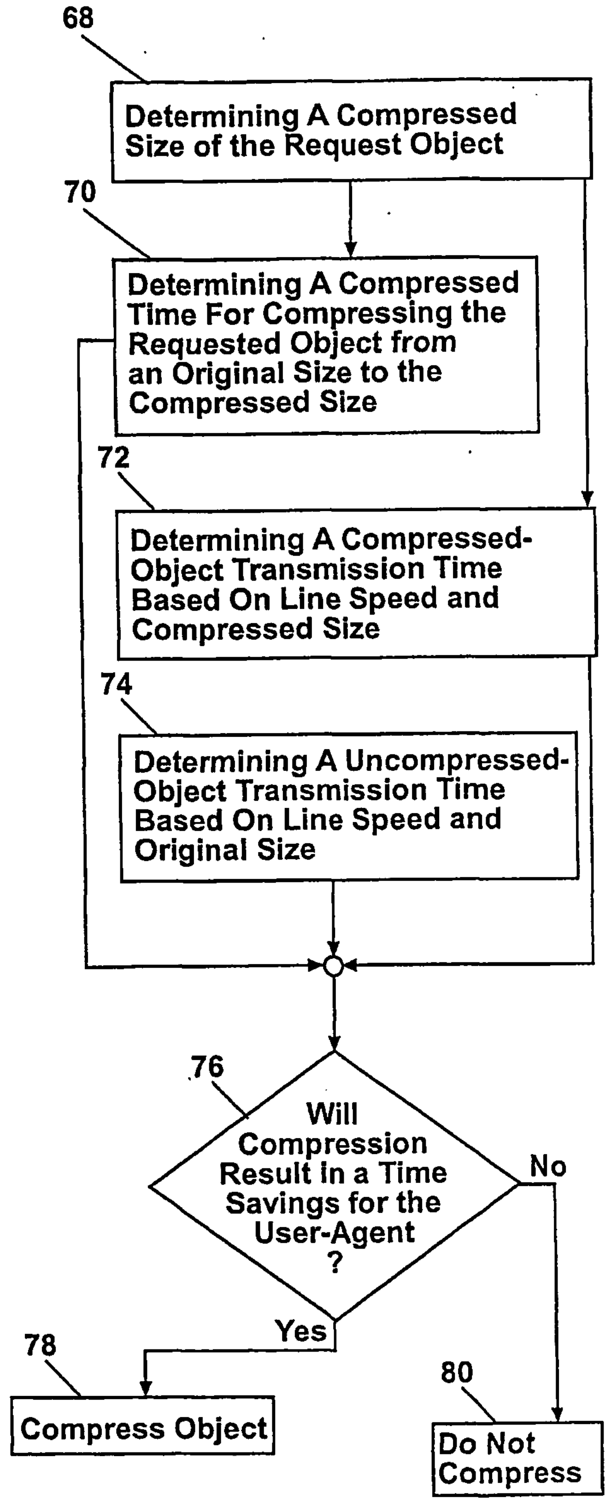
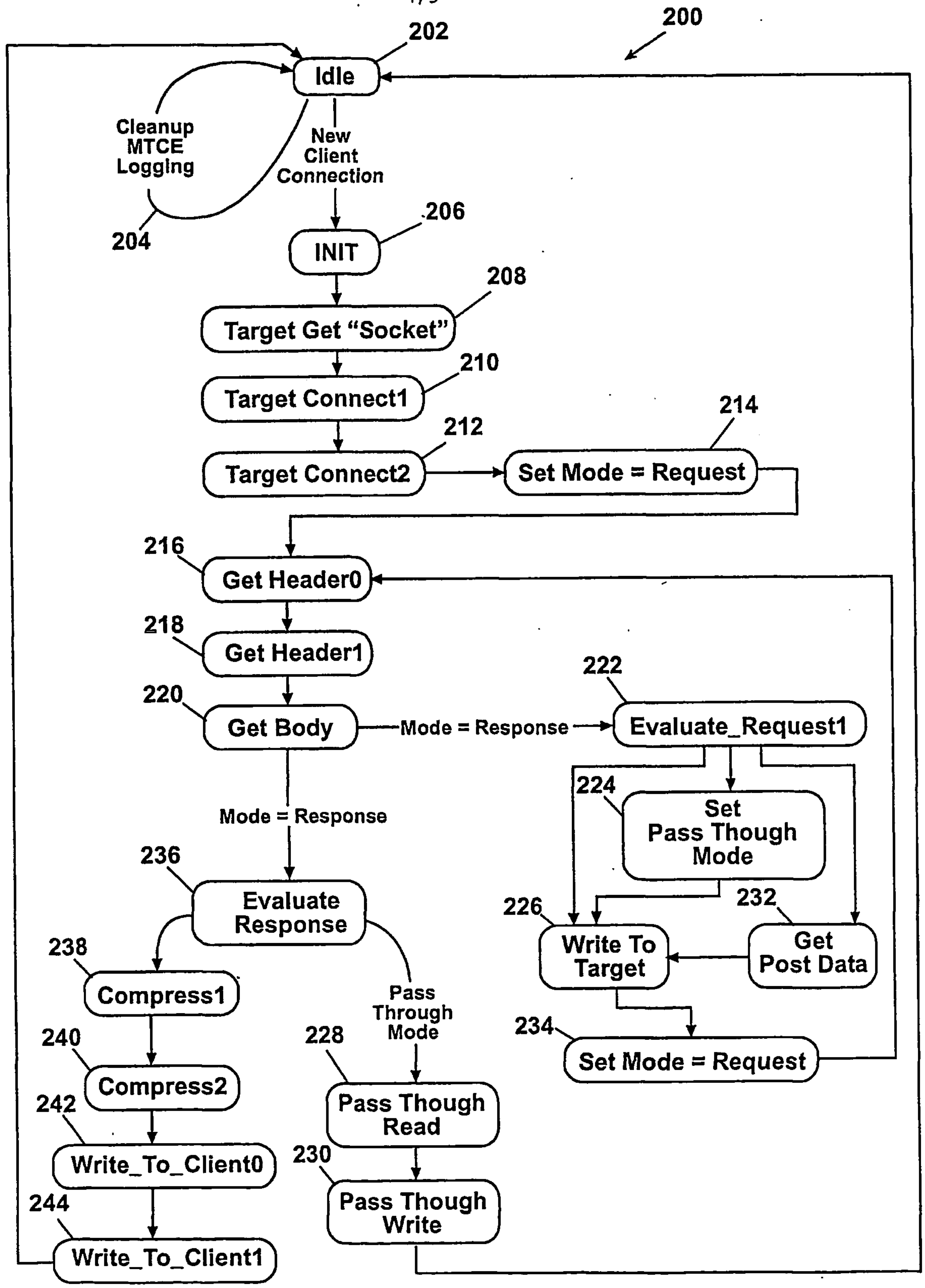
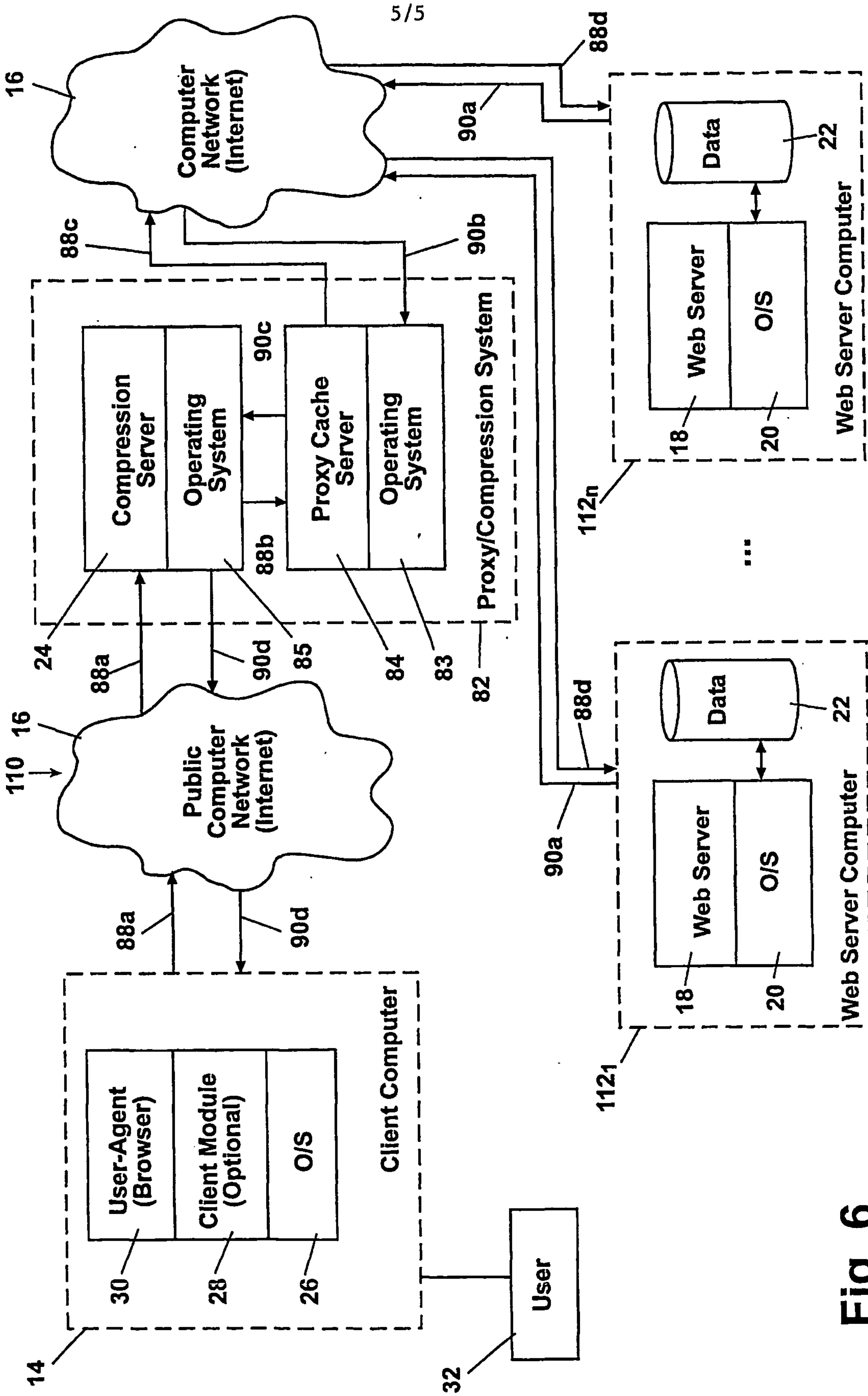

68
Determining A Compressed Size of the Request Object
70
Determining A Compressed Time For Compressing the Requested Object from an Original Size to the Compressed Size
72
Determining A Compressed-Object Transmission Time Based On Line Speed and Compressed Size
74
Determining A Uncompressed-Object Transmission Time Based On Line Speed and Original Size
76
Will Compression Result In a Time Savings for the User-Agent ?
No
Yes
78
Compress Object
80
Do Not Compress

Fig. 4



200
202
Idle
Cleanup
MTCE
Logging
204
New
Client
Connection
206
INIT
208
Target Get "Socket"
210
Target Connect1
212
Target Connect2
214
Set Mode = Request
216
Get Header0
218
Get Header1
220
Get Body
Mode = Response
222
Evaluate_Request1
224
Set
Pass Though
Mode
Mode = Response
236
Evaluate
Response
226
Write To
Target
232
Get
Post Data
238
Compress1
Pass
Through
Mode
234
Set Mode = Request
240
Compress2
228
Pass Though
Read
242
Write_To_Client0
230
Pass Though
Write
244
Write_To_Client1

Fig. 5



16
Computer Network (Internet)
88d
90a
Data
22
88c
90b
Compression Server
Operating System
90c
Proxy Cache Server
Operating System
88b
Proxy/Compression System
Web Server
O/S
Web Server Computer
18
20
112n
24
88a
90d
85
84
83
82
110
16
Public Computer Network (Internet)
88d
Data
22
90a
Web Server
O/S
Web Server Computer
18
20
1121
User-Agent (Browser)
Client Module (Optional)
O/S
Client Computer
30
28
26
14
User
32

Fig. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/23490

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 13/00
US CL : 709/203
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 707/10, 709/203, 217, 218, 231, 725/10

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	US 6,195,677 B1 (UTSUMI) 27 February 2001, col. 22, lines 1-31,	1-2
------	col. 24, lines 35-56;	3
Y,P	col. 27, lines 1-20;	11
	col. 25, lines 1-65.	12

		4-10, 13
Y	US 5,956,490 A (BUCHHOLZ et al.) 21 September 1999, col. 4,	
	lines 44-67, col. 5, lines 1-14;	4
	col. 6, lines 45-62.	5
Y,P	US 6,185,625 B1 (TSO et al.) 06 February 2001, col. 16, lines 7-67,	
	col. 17, lines 1-20;	6
	col. 15, lines 31-40;	7
	col. 17, lines 55-67, col. 18, lines 1-20;	8
	col. 18, lines 33-67.	9

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

- * Special categories of cited documents:
- "A" document defining the general state of the art which is not considered to be of particular relevance
- "E" earlier document published on or after the international filing date
- "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- "O" document referring to an oral disclosure, use, exhibition or other means
- "P" document published prior to the international filing date but later than the priority date claimed
- "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
- "&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
07 SEPTEMBER 2001	29 OCT 2001

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Peggy Harrod WALTER BENSON
Facsimile No. (703) 305-3230	Telephone No. (703) 305-4792

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/23490

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,P	US 6,175,856 B1 (RIDDLE) 16 January 2001, col. 2, lines 42-49.	10
Y	US 5,862,347 A (SUZUKI et al.) 19 January 1999col. 10, lines 27-67, col. 11, lines 1-67, col. 12, lines 1-34.	13